Magda's notes about papers read in **2022**

Last update: February 15, 2022

Informal notes for my future self who is likely to forget. I explain the papers the way I understand them, using terminology and logic natural to me. This means I may deviate from the original paper structure, notation, etc. At places, my interpretation my be incorrect due to lack of understanding. I will strive for this not to happen too often but I'm certainly not infallible.

This is a working document, not polished, with possible typos, editing errors, etc.

Imagery is usually taken directly from the original papers (unless otherwise stated).

Contents

1 graphNVP

Paper: Kaushalya Madhawa et al. “GraphNVP: an Invertible Flow-based Model for Generating Molecular Graphs”. 2019

ICLR not accept reviews Official code Local pdf

Read for Yoann’s work 5/2/2022 Notes taken: 6/2/2022

TLDR: Generate graphs $G = (A, X)$ through invertible flows. Dequantize A and X by uniform noise to make modelling continuous. Coupling strategy splitting the A and X matrices always by one node vs all the rest (breaks permutation invariance). The scale and translation in the coupling are graph NNs (do not have to be invertible) - for X functions of all other nodes and complete A matrix; for A edges between other nodes. Final reconstructed / generated A, X are quantized (floored) to get back to discrete.

Notes:

- Dequantize by simple uniform noise to get rid of the discreteness problem, then simply quantize (floor) to get back.
- ZINC-250K random sample of 250k molecules from complete ZINC to make experiments manageable.
- Good list of baseline models for experiments
- No validation checks during generation → versatile and fast.
- Coupling structure by nodes breaks permutation invariance - is this really a problem?

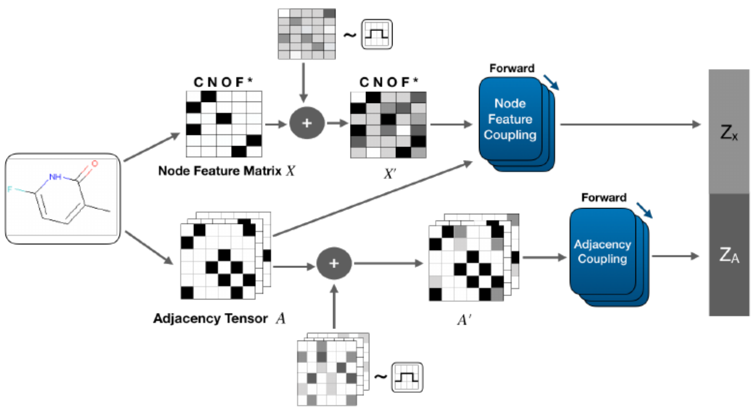


Figure 1: For the forward transform, A and X matrices are perturbed by uniform noise. The node coupling is conditioned on the A matrix.

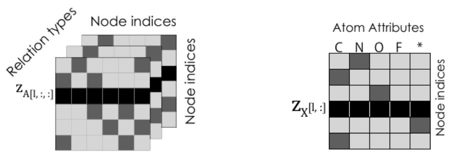


Figure 2: The coupling structure splits the A and X matrices as one node vs all the rest. This works best in experiments, though breaks down permutation invariance.

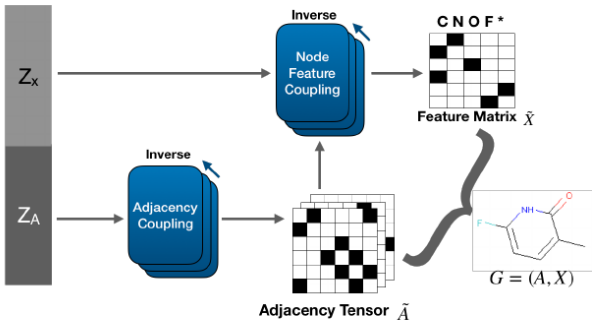


Figure 3: Generation is in two steps because the reverse coupling transform of X is conditioned on A so this needs to be generated first. Using simple dequantization (flooring) to get from continuous flow output to discrete A, X.

Experiments: on QM9 and ZINC-250K with kekulized graphs (without hydrogens) - our *cheat* version. Have not so great validity but high uniqueness. Claim benefits in speed of generation as opposed models that have complex decoders checking the validity of molecules in the process. Some comments on smoothness of latent space and possibility to interpolate for property control but not very convincing.

2 MPGVAE

Paper: Daniel Flam-Shepherd et al. "MPGVAE: improved generation of small organic molecules using message passing neural nets". In: *Machine Learning: Science and Technology* 2.4 (2021). Publisher: IOP Publishing, p. 045010

no reviews no official code Local pdf

Read for Yoann's work 5/2/2022 Notes taken: 6/2/2022

TLDR: Builts on GraphVAE (Simonovsky and Komodakis, 2018) but bypasses the graph matching problem by relaying on message passing NN in encoder and decoder so that the reconstruction loss can be calculated directly. Message passing over edges, nodes use attention to aggregate these. To get from node representation to laten vector they use set2vec from Vinyals 2015 to be invariant to the node ordering. To get from latent back to graph they use simply linear layer and then message passing, finishing with softmax over node and egde representation to create stochastic edge and node graph representations from which the graphs can be samples (in practice assume by max category). The reconstruction loss assumes independence of node and edge features $p\theta(G|z) = \Pi_{v \in G} p(x_v|z) \Pi_{u \in G} p(e_{uv}|z)$

Notes:

- code not publicly available - asked first author, no answer
- set2vec to be invariant to node ordering when getting from the node representation in the last MPNN layer the latent representation z, on the way back can use use MLP and MPNN cause the ordering problem was solved when moving to the latent
- ***I still do not see how the reconstruction loss can be evaluated. Even with the independence assumptions, each of the node/ edge likelihoods has to be matched to the respective edge/ node in the original graph. How can this be done without some matching procedure?***

Graph with adjacency matrix, edge feature tensor and node feature matrix $G = (A, E, X)$. Assume independence between node and edge probability distributions (Erdos-Rényi graph model) such as in "GraphVAE" of Simonovsky and Komodakis (2018).

Starting point is standard VAE and ELBO. Both edge and node features are categorical. Build a message passing NN (MPNN) into encoder and decoder of the VAE.

Refers to Graphite from Grover et al 2018 as nearest baseline - how different is this?

Claim that has lower complexity than GrpahVAE (how so?).

Encoder: Use MPNN to map node and edge representations into latent space. Messages are constructed at edge level as linear functions of itself, neighbouring nodes, and edge features passed through tanh nonlinearity. Edge updates are directly the messages. Node updates first use attention to reweight the messages and then pass it through a GRUCell (this is not very clear, there is something strange in eq. 11 - should be a message for u only? not uv?) Use set2set from Vinyals 2015 (unlike in seq2seq, in sets the order should not matter and the model should produce the same representation in arbitrary ordering of the set) to read from last node representation to fixed-sized hidden representation z.

Decoder: Pass latent through linear layer to expand dims and then RNN to get initial node featurs (edge features zero - unconnected graph). MPNN identical to encoder - update both nodes and eges. Readout the last layer into stochastic graph representation by passing it through single layer NN and softmax.

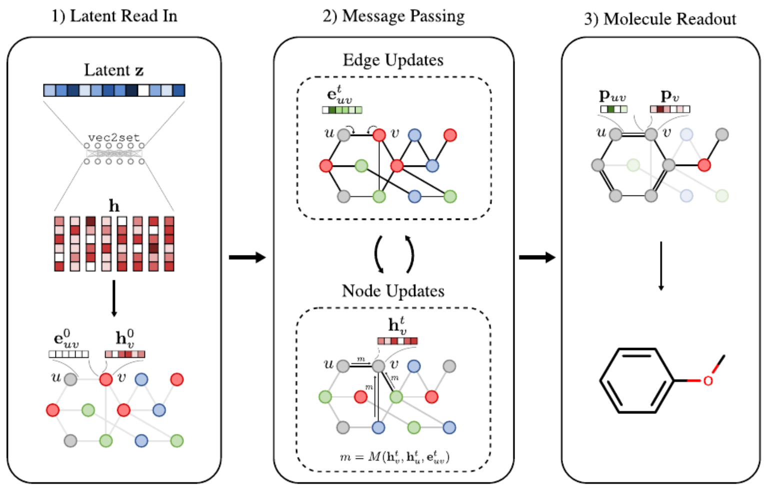


Figure 4: 1) Read in latent z by passing it through linear layer to get to correct node dimensions and pass it through RNN to get initial node featues. Edge features fixed to zero - unconnected graph. 2) Message passing on both, node and edge representations. 3) Use last node and edge representation to predict independent node and edge feature probabilities.

Experiments: Over QM9 (not clear whether kekulized or not and if has hydrogens) - best validity compared to GrphVAE, MolGAN, CharacterVAE and Grammar VAE. Generate 10^4 molecules and achieve validity over 90%, unique almost 70% and novel about 50% - much better then baseline. Also evaluates the continuous (chemical properties) and discrete (avg number of atom types and ring size) property distributions of generated vs training sample graphs - claim to have both better than baseline.

3 FMTDA

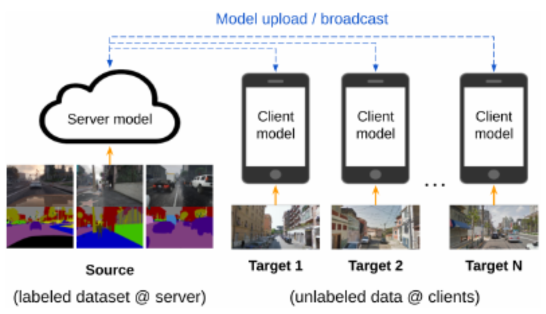
Paper: Chun-Han Yao et al. “Federated multi-target domain adaptation”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision.* 2022, pp. 1424–1433

reviews not available   No code   Local pdf

Presented by Manuel in CAIRO readings on 9/2/2022   Notes taken: 13/2/2022

TLDR: Federated learning (FL) setup where server has labelled data and clients have only unlabelled data moreover with shifted data domains. Builds on maximum classifier discrepancy (MCD) [39] training which is a kind of adversarial game with minimax loss where there are two classifiers acting as discriminators and a feature extractor acting as a generator. The generator tries to align the features between the source and target domains and the discriminator step uses two classifiers that try to disagree as much as possible on the target domain while being correct for the source domain. In the FL setup they discuss here the problem is all about what is trained where and how it is communicated. Feature extractor needs to be trained on the server together with a global classifier while local classifiers are trained on the clients. The data domains are modelled by a simple GMM which is used to re-weight examples when calculating the classification losses for within and out of distribution data examples.

Notes:
- Nice paper as not so trivial intro to FL
- Experiments still surprisingly low accuracy - 30% on MNIST like data.
- Possible future work: take on board new clients with new target domains quickly (cannot retrain the whole system every time there is a new client). Use GMMs to find the nearest domain and initialize new client by this classifier. Adapt only client and server and gradually spread?



Federated learning (FL) setup in which the server has labelled data and the clients have unlabelled data. Moreover, the data domains on the server (source domain D_S) and the clients (target domains D_T) are assumed to be not (perfectly) aligned.

Model aggregation on the server in this case won’t work because of inter-client discrepancies. Instead they propose dual adaptation approach whereby the models are adapted both at the server and the clients. The models are based on a common feature extractor G and classifier heads F_i.

They reuse the maximum classifier discrepancy (MCD) approach introduced in [34 ref in the paper] which has similar minimax game as GAN.

$$\begin{aligned}\text{Discriminator loss:} \quad & \arg\min_{F_1,F_2} \mathcal{L}_{ce}(D_S) - \mathcal{L}_{adv}(D_T) \\ \text{Generator loss:} \quad & \arg\min_{G} \mathcal{L}_{adv}(D_T) \ ,\end{aligned}$$

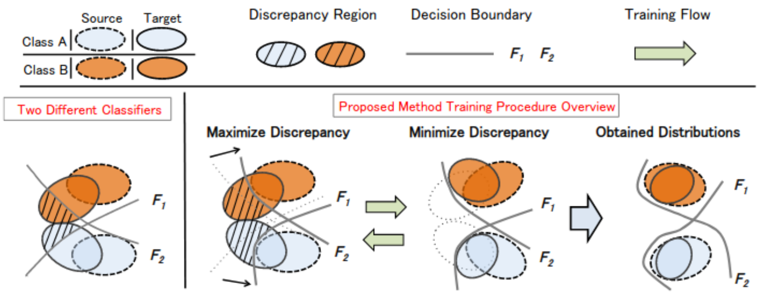
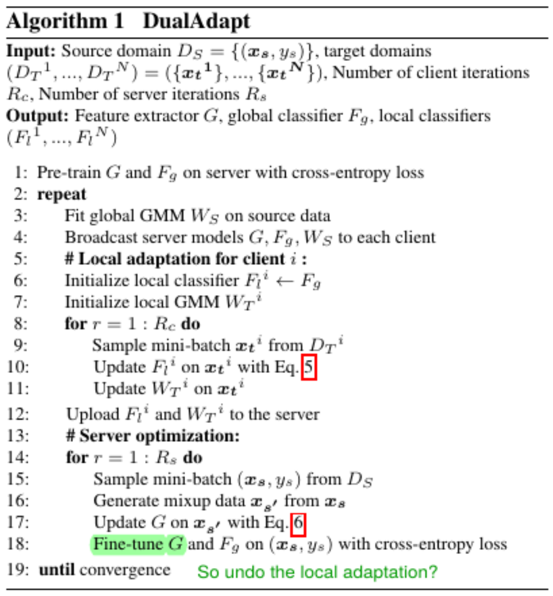


Figure 5: 34 MCD: On one hand train 2 classifiers F_1 and F_2 that perform well on the source domain but at the same time maximize the classification discrepancy (discriminators) in the target domain. On the other train the feature extractor (generator) to fool the discriminators, that is minimize the discrpancy. This shall push the target features into the source domain.

where $\mathcal{L}_{ce}(D_S)$ is standard cross-entropy loss over the labelled source domain and is a suitable discrepancy loss such as L^1 norm that they use here $\mathcal{L}_{adv}(D_T)\mathbb{E}_{\mathbf{x}\sim D_T}\|F_1(G(\mathbf{x})) - F_2(G(\mathbf{x}))\|_1$.



```
Algorithm 1  DualAdapt
Input: Source domain $D_S = \{(\boldsymbol{x_s}, y_s)\}$, target domains
$({D_T}^1, ..., {D_T}^N) = (\{\boldsymbol{x_t}^1\}, ..., \{\boldsymbol{x_t}^N\})$, Number of client iterations
$R_c$, Number of server iterations $R_s$
Output: Feature extractor $G$, global classifier $F_g$, local classifiers
$({F_l}^1, ..., {F_l}^N)$

 1: Pre-train $G$ and $F_g$ on server with cross-entropy loss
 2: repeat
 3:     Fit global GMM $W_S$ on source data
 4:     Broadcast server models $G, F_g, W_S$ to each client
 5:     # Local adaptation for client $i$ :
 6:     Initialize local classifier ${F_l}^i \leftarrow F_g$
 7:     Initialize local GMM ${W_T}^i$
 8:     for $r = 1 : R_c$ do
 9:         Sample mini-batch ${\boldsymbol{x_t}}^i$ from ${D_T}^i$
10:         Update ${F_l}^i$ on ${\boldsymbol{x_t}}^i$ with Eq. 5
11:         Update ${W_T}^i$ on ${\boldsymbol{x_t}}^i$
12:     Upload ${F_l}^i$ and ${W_T}^i$ to the server
13:     # Server optimization:
14:     for $r = 1 : R_s$ do
15:         Sample mini-batch $(\boldsymbol{x_s}, y_s)$ from $D_S$
16:         Generate mixup data $\boldsymbol{x}_{s'}$ from $\boldsymbol{x_s}$
17:         Update $G$ on $\boldsymbol{x}_{s'}$ with Eq. 6
18:         Fine-tune $G$ and $F_g$ on $(\boldsymbol{x_s}, y_s)$ with cross-entropy loss
19: until convergence      So undo the local adaptation?
```

In their setup they update the generator (feature extractor) G on the server and have a global classifier F_g on the server and local classifiers F_t^i on the clients. They train the local classifiers f_t^i via the discriminator loss by maximizing the $\mathcal{L}_{adv}$ discrepancy between the global (fixed here) and local classifiers together with minimizing the cross-entropy using the predictions of the global classifier F_g as the ground truth. Since the labels from F_g may not be perfect, they use GMM fitted on D_S to re-weight the target examples (if out of source distribution, should have lower weight).

They also fit GMM on the clients. These local GMMs are uploaded together with the local classifiers to the server. The feature extractor G is trained on the server by minimizing the generator loss with $\mathcal{L}_{acv}$ evaluated over mix-up instances (simple avg of two randomly sampled instances from D_S) weighted by local GMM mixtures. This is done for each target domain so that the server loss is the average across all the target models.

At inference they classify new instances by averaging the global and local classifier predictions.

They show experiments on MNIST as the source domain and MNIST-M, SVHN, USPS etc. as targets; DomainNet (clipart, infograph, painting ..)and GTA5 dataset with street-views simulated from computer games. It seems to perform better then baselines though still only about 30% accuracy on MNIST-like data.

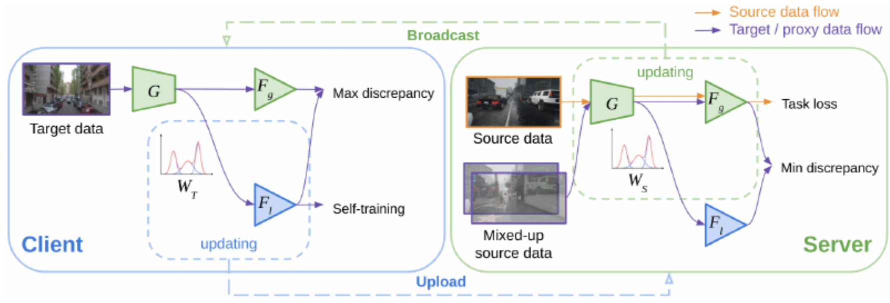


Figure 6: Dual model adaptation on both clients and server.

4 Diffusion models

Paper: Jonathan Ho et al. "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems.* Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 6840–6851

reviews code Local pdf

Read with Simon on 27/1/2022 Notes taken: 13/2/2022

TLDR: Diffusion process is a Markov chain gradually adding small quantities of noise to the data until the signal is completely destroyed - seems as random noise. The diffusion model is trained to reverse this process. When the diffusion consists of Gaussian noise, the sampling transitions can be Gaussian as well. The diffusion (forward) process is seen as a posterior $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ of the reverse model $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Here, however, the posterior is fixed (Gussians with variance schedule) so that the full chain $\mathbf{x}_{1:T}$ can be generated from the data $\mathbf{x}_0$ and the model is then trained to get back using variational lower bound on $\log p_\theta(\mathbf{x}_0)$ as the loss. The trick is in rewriting the lower bound so that it is trainable. It all relies on tractability of the conditionals because they are all Gaussians.

Notes:
- What if the Markov chain was not Gaussian? Would it still be possible to find a tractable solution and would it be good for anything? Probably not. The initial generative noise would then be non-Gaussian - so what?

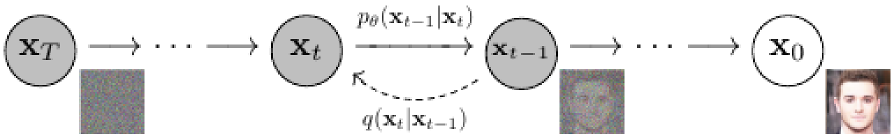

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 7: Forward diffusion process q and reverse diffusion model p.

Diffusion model is latent variable model $p_\theta(\mathbf{x}_0) = \int p_\theta(\mathbf{x}_{0:T})d\mathbf{x}_{1:T}$ where the latents $\mathbf{x}_{1:T}$ have the same dims as the data $\mathbf{x}_0 \sim q(\mathbf{x}_0)$. The joint distribution called the **reverse process** is a **Markov chain with learned Gaussian transitions**

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\Pi_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \qquad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$$

The diffusion process (**forward process**) is also a Markov chain adding gradually Gaussian noise to the data and is the approximate posterior of the model

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \Pi_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1}), \qquad q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \boldsymbol{I}) \ ,$$

where $\beta_1, \ldots, \beta_T$ is a variance schedule.

Training is done by maximizing the usual variational lower bound on $\log p_\theta(\mathbf{x}_0)$ with the whole $\mathbf{x}_{1:T}$ as latent and using the Markov chain properties to decompose it into a sum of KL divergences over the chain conditionals $q(\mathbf{x}_t|\mathbf{x}_{t-1})$. β's can be trained by reparametrization trick or fixed as hyperparameters.

Interestingly, due to the specific formulation of the forward process, any timestep t can be sampled using only the initial point $\mathbf{x}_0$ for conditioning

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, 1-\bar{\alpha}_t\boldsymbol{I}), \qquad \alpha_t = 1-\beta_t,\ \bar{\alpha}_t = \Pi_{s=1}^t \alpha_s$$

Due to this, the forward process Markov chain conditionals (the posteriors) are tractable when conditioned on $\mathbf{x}_0$

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \bar{\mu}(\mathbf{x}_t, \mathbf{x}_0), \bar{\beta}_t\boldsymbol{I}) \ ,$$

where $\bar{\mu}$ is a fixed simple function of $\mathbf{x}_t, \mathbf{x}_0$ and some αs and βs (see equation 7 in paper for details).

In their setup they do not learn βs but fix them - approximate posterior has no trainable parameters (no θs in the above), forward process is fixed.

The reverse process $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$ has in principal trainable mean and variance functions but they fix the variance to isotropic $\sigma_t\boldsymbol{I}$ (σ_t is a simple fixed function of β_t - see paper). With these the KL divergences in the ELBO loss can be evaluated in closed form (boil down to ℓ_2 norms) and the only thing we learn are the mean functions $\mu_\theta(\mathbf{x}_t, t)$ of the $t-1$ step in the reverse process.

We can further reparametrize the sampling of x_t by sampling the standard normal $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$ and $x_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ and hence $\mathbf{x}_0 = (\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon)/\sqrt{\bar{\alpha}_t}$. This does not seem very useful cause we do not learn any of the parameters. However, since by the loss the mean of the reverse process at step t shall be the same as the mean of the forward process (which is obtainable in closed from from αs and βs and the actual values of $\mathbf{x}$ generated by the forward process), we get that

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(\mathbf{x}_t, t)) \ ,$$

where $\epsilon_\theta(\mathbf{x}_t, t)$ is the function that shall be trained to minimize the ELBO. To sample, we use $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(\mathbf{x}_t, t)) + \sigma_t\mathbf{z},\ \mathbf{z} \sim \mathcal{N}(0, \boldsymbol{I})$ recursively starting from $\mathbf{x}_t \sim \mathcal{N}(0, \boldsymbol{I})$ all the way back to $\mathbf{x}_0$.

With the introduction of ϵ the loss now resembles denoising score matching of Song and Ermon 2020 and the sampling process the Langevin dynamics.

Nice experiments.

5 Score-based models

Paper: Yang Song and Stefano Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution". In: *arXiv:1907.05600 [cs, stat]* (2020)

reviews code Local pdf

Read with Simon on 9/2/2022 Notes taken: 14/2/2022

TLDR: Generative model where data are generated through Langevin dynamics (Welling and Teh 2011) using gradients estimated through score matching (Hyvarinen 2005). They perturb the data with small Gaussian noise to ensure the data don't lie on a low dimensional manifold within the original feature space which would make the gradients ill-defined. They also propose to anneal the Langevin dynamics sampling process with gradually decreasing noise levels.

Notes:
- Mixing a lot of things together into something that in the end works
- A bit of archeology to dig up score mathchin and Langevin dynamics sampling
- The random projections idea is here nonsensical I think, it is a different setup. They put it just to increase citation score? ;)
- The perturbation story holds in retrospect but I wonder how did they come up with it in the first place.

The model is based on estimating the (Stein) ***score that is the gradient of the log data density with respect to the data*** $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ (though standard score function in statistics is the gradient with respect to the parameters of the distribution at a fixed data point - this is acknowledged in the original [33] paper of Jordan's group but not in this paper).

The score network $s_\theta : R^D \to R^D$ shall be trained to estimate the score without estimating the data likelihood $p(\mathbf{x})$ first. Once trained, it can be used within Langevin dynamics sampling to generate new data examples.

The **score matching** objective is the minimization of $0.5\mathbb{E}_{p_{data}} \|s_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p(\mathbf{x})\|_2^2$ which has been show (in Hyvarinen 2005 to be equivalent to the minimization of

$$\mathbb{E}_{p_{data}}(\mathbf{x}) \left[tr(\nabla_{\mathbf{x}} s_\theta(\mathbf{x})) + \frac{1}{2} \|s_\theta(\mathbf{x})\|_2^2 \right]$$

This is important cause it removes the unknown data density $p(\mathbf{x})$ from the loss. The problem is, however, that it does not scale to high dimensions because of the need to get the trace of the jacobian. *Because of the jacobian?*

Instead, they propose to use the **denoising score matching**

$$\frac{1}{2}\mathbb{E}_{q_\sigma(\widetilde{\mathbf{x}}|\mathbf{x})} \left[\|s_\theta(\widetilde{\mathbf{x}}) - \nabla_{\widetilde{\mathbf{x}}} \log q_\sigma(\widetilde{\mathbf{x}}|\mathbf{x})\|_2^2 \right]$$

which is shown in Vincent 2011 to be equivalent to score matching with the marginal $q_\sigma(\widetilde{\mathbf{x}}) = \int q_\sigma(\widetilde{\mathbf{x}}|\mathbf{x}) p_{data}(\mathbf{x}) d\mathbf{x}$ *Not very intuitive but I trust the proof.* In turn this would also be equivalent to score matching with the original data density p_{data} if the noise is very little so that $q_\sigma(\mathbf{x}) \approx p_{data}(\mathbf{x})$.

Finally, they claim by the use of **random projections** they can reduce the complexity of the trace jacobian calculation recognized above.

$$\mathbb{E}_{p_v} \mathbb{E}_{p_{data}} \left[\mathbf{v}^T \nabla_{\mathbf{x}} s_\theta(\mathbf{x}) \mathbf{v} - \frac{1}{2} \|s_\theta(\mathbf{x})\|_2^2 \right]$$

I don't see how this helps cause you still need to get the jacobian. Hm ...

Once they have trained the score estimator $s_\theta(\mathbf{x}) \approx \nabla_\mathbf{x} \log p(\mathbf{x})$ they can use **Langevin dynamics** as explained in Welling and Teh 2011 to sample examples

$$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_{t-1} + \frac{\epsilon}{2} s_\theta(\hat{\mathbf{x}}_{t-1}) + \sqrt{\epsilon}\mathbf{z}_t \ ,$$

where $\mathbf{z}_t \sim \mathcal{N}(0, \boldsymbol{I})$ is the noise, $\hat{\mathbf{x}}_0$ sampled from a prior $\pi(\mathbf{x})$ and step size $\epsilon > 0$. When $\epsilon \to 0$ and $T \to \infty$ then $\hat{\mathbf{x}}_T$ has the same distribution as the desired $p(\mathbf{x})$.

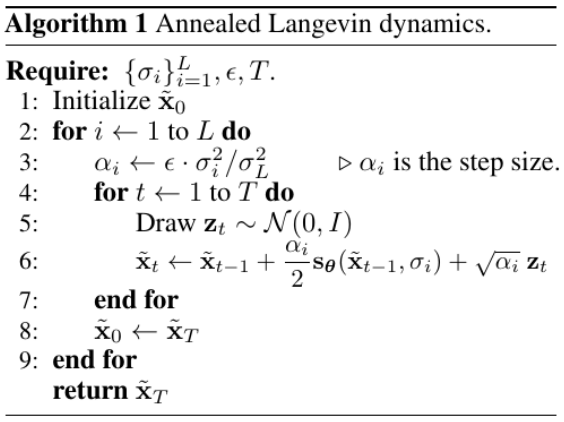
This is a kind of gradient ascent with noise. I can see that we gradually move the samples to a higher density region and adding a little bit of noise while doing it. How come that in infinity with decreasing noise all the $\hat{\mathbf{x}}_T$ do not end in the mode of the distribution? I guess Welling and Teh 2011 has the answer :).

They next claim that because the data may reside on a *low dimensional manifold*, the gradient of the density in the original feature space may be ill-defined. Furthermore, the model struggles at *low density regions*. It is difficult to estimate the score correctly in these regions due to undersampling. Similarly, Langevin dynamics cannot balance mixtures of distributions correctly. They solve these by perturbing the data with various levels of noise σ (a decreasing schedule) and then estimating the corresponding scores by training a score network conditioned on the noise level $s_\theta(\mathbf{x}, \sigma)$. This naturally leads back to the denoising score matching version of the loss optimization which moreover with Gaussian perturbations $q_\sigma(\widetilde{\mathbf{x}}|\mathbf{x}) \sim \mathcal{N}(\mathbf{x}, \sigma\boldsymbol{I})$ nicely simplifies to

$$l(\theta, \sigma) = \frac{1}{2}\mathbb{E}_{p_{data}(\mathbf{x})}\mathbb{E}_{q_\sigma(\widetilde{\mathbf{x}}|\mathbf{x})}\left[\|s_\theta(\widetilde{\mathbf{x}}) - \frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma^2}\|_2^2\right]$$

$$\mathcal{L}(\theta) = \frac{1}{L}\sum_i^L \lambda(\sigma_i) l(\theta, \sigma_i)$$

so that the final loss is a weighted average of the noisy score losses. They use trivially $\lambda(\sigma_i) = \sigma_i^2$.


Algorithm 1 Annealed Langevin dynamics.

```
Require: $\{\sigma_i\}_{i=1}^L, \epsilon, T$.
1: Initialize $\tilde{\mathbf{x}}_0$
2: for $i \leftarrow 1$ to $L$ do
3:     $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$        ▷ $\alpha_i$ is the step size.
4:     for $t \leftarrow 1$ to $T$ do
5:         Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
6:         $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_\theta(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$
7:     end for
8:     $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
9: end for
   return $\tilde{\mathbf{x}}_T$
```

They then use the noisy estimates of the scores in the Langevin dynamics sampling process - starting from the score estimated from the most noisy data generate a sample through multiple steps of the Langevin dynamics update, then use this sample as the initial point for next iteration of the updates with lower noise. The step size is also gradually decreasing by a fixed schedule according to the noise levels.

Nice experiments showing how the sampled data gradually get cleaner through the annealed Langevin dynamic sampling.

References

[1] Daniel Flam-Shepherd, Tony C. Wu, and Alan Aspuru-Guzik. "MPGVAE: improved generation of small organic molecules using message passing neural nets". In: *Machine Learning: Science and Technology* 2.4 (2021). Publisher: IOP Publishing, p. 045010 (cit. on p. 4).

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems.* Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 6840–6851 (cit. on p. 9).

[3] Aapo Hyvarinen. "Estimation of Non-Normalized Statistical Models by Score Matching". In: (2005), p. 15 (cit. on p. 11).

[4] Kaushalya Madhawa et al. "GraphNVP: an Invertible Flow-based Model for Generating Molecular Graphs". 2019 (cit. on p. 2).

[5] Martin Simonovsky and Nikos Komodakis. "GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders". In: *arXiv:1802.03480 [cs]* (2018) (cit. on p. 4).

[6] Yang Song and Stefano Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution". In: *arXiv:1907.05600 [cs, stat]* (2020) (cit. on pp. 10, 11).

[7] Pascal Vincent. "A Connection Between Score Matching and Denoising Autoencoders". In: *Neural Computation* 23.7 (2011). Conference Name: Neural Computation, pp. 1661–1674 (cit. on p. 11).

[8] Max Welling and Yee Whye Teh. "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: (2011), p. 8 (cit. on pp. 11, 12).

[9] Chun-Han Yao et al. "Federated multi-target domain adaptation". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision.* 2022, pp. 1424–1433 (cit. on p. 6).

Index